Comprehensive Guide to Git Branching Strategy

Git branching strategies help organize the development, testing, and deployment processes by isolating code changes in different branches. A well-structured strategy is crucial for managing environments like **Development (Dev)**, **Quality Assurance (QA)**, **Pre-Production (PPD)**, **Production (Prod)**, and **Disaster Recovery (DR)**.

This guide covers:

1. **Branching Strategy with Dev, QA, PPD, Prod, and DR**
2. **Branching in Feature, Bugfix, and Hotfix Workflows**
3. **Code Promotion and Pull Requests**
4. **Disaster Recovery Explained**
5. **Branching Strategy for QA and Prod**

---

1. Branching Strategy for Dev, QA, PPD, Prod, and DR

Branch Structure

Branch	**Purpose**
main (or prod)	Represents the production-ready code deployed to end users.
dr	Backup branch identical to prod, used for disaster recovery.
ppd	Pre-production testing branch for final validation before deployment.
qa	Quality Assurance branch for testing integrated features from dev.
dev	Integration branch where all feature branches are merged and tested first.
feature/<name>	Short-lived branches for individual feature development.
bugfix/<name>	Short-lived branches for fixing bugs identified in testing or production.
hotfix/<name>	Emergency fixes for production issues.

Workflow Overview

1. **Feature Development**:
    - Developers create feature/<name> branches from dev to work on new features.
    - After completing development, the feature branch is merged into dev.
2. **Bug Fixes**:
    - bugfix/<name> branches are created from dev (or main for critical issues).
    - Fixes are merged into dev and tested in QA.

3. **Quality Assurance (QA)**:
    - Code from dev is merged into qa for dedicated QA testing.
    - Testers validate new features and fixes in the qa branch.
4. **Pre-Production (PPD)**:
    - After QA approval, code from qa is merged into ppd for final testing.
    - The PPD environment simulates production settings.
5. **Production Deployment**:
    - After PPD approval, code is merged into main (or prod) for deployment.
    - A copy of main is maintained in the dr branch for disaster recovery.

Example Workflow

1. **Create a feature/login branch:**

```
git checkout -b feature/login dev
```

2. **Develop, commit, and merge to dev:**

```
git commit -m "Add login functionality"
git checkout dev
git merge feature/login
git push origin dev
```

3. **Merge dev into qa for testing:**

```
git checkout qa
git merge dev
git push origin qa
```

4. **After QA testing, merge qa into ppd:**

```
git checkout ppd

git merge qa

git push origin ppd
```

5. **After PPD testing, merge ppd into main (prod):**

```
git checkout main
git merge ppd
git push origin main
```

6. **Sync main with dr:**

```
git checkout dr
git merge main
git push origin dr
```

---

2. Feature, Bugfix, and Hotfix Branches

Feature Branches

- Purpose: Develop new features in isolation.
- Naming Convention: feature/<name>.

Workflow:

1. Create a branch:

```
git checkout -b feature/login dev
```

2. Develop and commit:

```
git add .
git commit -m "Add login functionality"
```

3. Merge into dev:

```
git checkout dev
git merge feature/login
git push origin dev
```

Bugfix Branches

- Purpose: Fix bugs identified in dev, qa, or prod.
- Naming Convention: bugfix/<name>.

Workflow:

1. Create a branch:

```
git checkout -b bugfix/fix-login dev
```

2. Fix the bug and commit:

```
git add .
git commit -m "Fix login validation issue"
```

3. Merge into dev or qa:

```
git checkout dev
git merge bugfix/fix-login
git push origin dev
```

Hotfix Branches

- Purpose: Address critical issues in production immediately.
- Naming Convention: hotfix/<name>.

Workflow:

1. Create a branch:

```
git checkout -b hotfix/fix-login main
```

2. Fix the bug and commit:

```
git add .
```

```
git commit -m "Fix critical login issue"
```

3. Merge back into main and dev:

```
git checkout main
git merge hotfix/fix-login
git push origin main
```

```
git checkout dev
git merge hotfix/fix-login
git push origin dev
```

---

3. Code Promotion, Pull Requests, and Merging

Code Promotion

Code promotion refers to moving changes from lower environments (Dev, QA) to higher environments (PPD, Prod).

Promotion Flow:

1. Merge dev into qa for QA testing.
2. Merge qa into ppd after QA approval.
3. Merge ppd into main (prod) after final validation.

Automating Promotion:

- Use CI/CD tools like Jenkins, GitHub Actions, or GitLab CI to automate code promotion through pipeline stages.

Pull Requests (PR)

Pull requests are used to review and merge changes from one branch to another.

Workflow:

1. Developer creates a pull request from feature/<name> to dev.
2. Code is reviewed by peers.
3. Once approved, the PR is merged.

Example:

1. Push a feature branch:

```
git push origin feature/login
```

2. Create a PR on GitHub to merge feature/login into dev.
3. After approval, merge the PR.

---

4. Disaster Recovery (DR) Branch

Purpose

The DR branch serves as a backup of the main branch. It ensures that the latest stable code is readily available for redeployment during critical failures.

Workflow

1. Sync main with dr after every deployment:

```
git checkout dr
git merge main
git push origin dr
```

2. If a disaster occurs, redeploy from dr:

```
git checkout dr
git pull origin dr
# Redeploy using CI/CD or manual methods
```

Best Practices

- Automate syncing between main and dr.
- Regularly test the DR deployment process.

---

5. Branching Strategy for QA and Prod

For projects with only **QA** and **Prod** environments, the strategy simplifies to:

- main (Prod): Stable production code.
- qa: Testing environment.

Workflow:

1. Develop in qa:

```
git checkout -b feature/login qa
```

2. Merge completed features into qa:

```
git checkout qa
git merge feature/login
git push origin qa
```

3. After QA testing, merge qa into main for deployment:

```
git checkout main
git merge qa
git push origin main
```

Best Practices for Git Branching Strategy

1. **Use Naming Conventions**:
    - feature/<name>, bugfix/<name>, hotfix/<name> for clarity.
2. **Automate Testing and Deployment**:
    - Use CI/CD pipelines to enforce code quality and automate promotions.
3. **Regular Syncing**:
    - Frequently merge changes from lower branches (dev) into higher branches (qa, ppd) to avoid conflicts.
4. **Code Reviews**:
    - Mandate pull requests and code reviews for every merge.
5. **Document the Process**:
    - Clearly define branching rules and workflows for the team.

By following this detailed branching strategy, teams can efficiently manage multiple environments, streamline feature development, and ensure robust disaster recovery.